RS data

Z8 Applications

Introduction

The Z8601 is a single-chip microcomputer with four 8-bit I/O ports, two counter/timers with associated prescalers, asynchonous serial communication interface with programmable baud ranges, and sophisticated interrupt facilities. The Z8601 can access data in three memory spaces: 2K bytes of on-chip ROM and 62K bytes of external program memory, 144 bytes of on-chip Register, and 62K bytes of external data memory.

The Z8671 is a Z8601 with a Basic/Debug Interpreter and Debug monitor preprogrammed into the 2K bytes of on-chip ROM. This application note discusses some considerations in designing a low-complexity board that runs the Basic/Debug Interpreter and Debug monitor with an external 4K bytes of RAM and 2K bytes of ROM. The board stands alone, allowing users to connect it with a terminal via an RS232 connector and run the Basic/Debug Interpreter.

The user of this board can run Basic/Debug with little knowlege of the Z8601. The board, however, derives its power through its ability to execute assembly language programs. To use the board to its full potential, the Z8 Technical Manual (902-063) and the Z8 Assembly Language Programming Manual (902-041) should be read. The Z8671 Basic/Debug Reference Manual (902-057) provides general information, statement syntax, memory allocations, and other material regarding Basic/Debug and the Debug monitor provided by the Z8671.

Basic/Debug

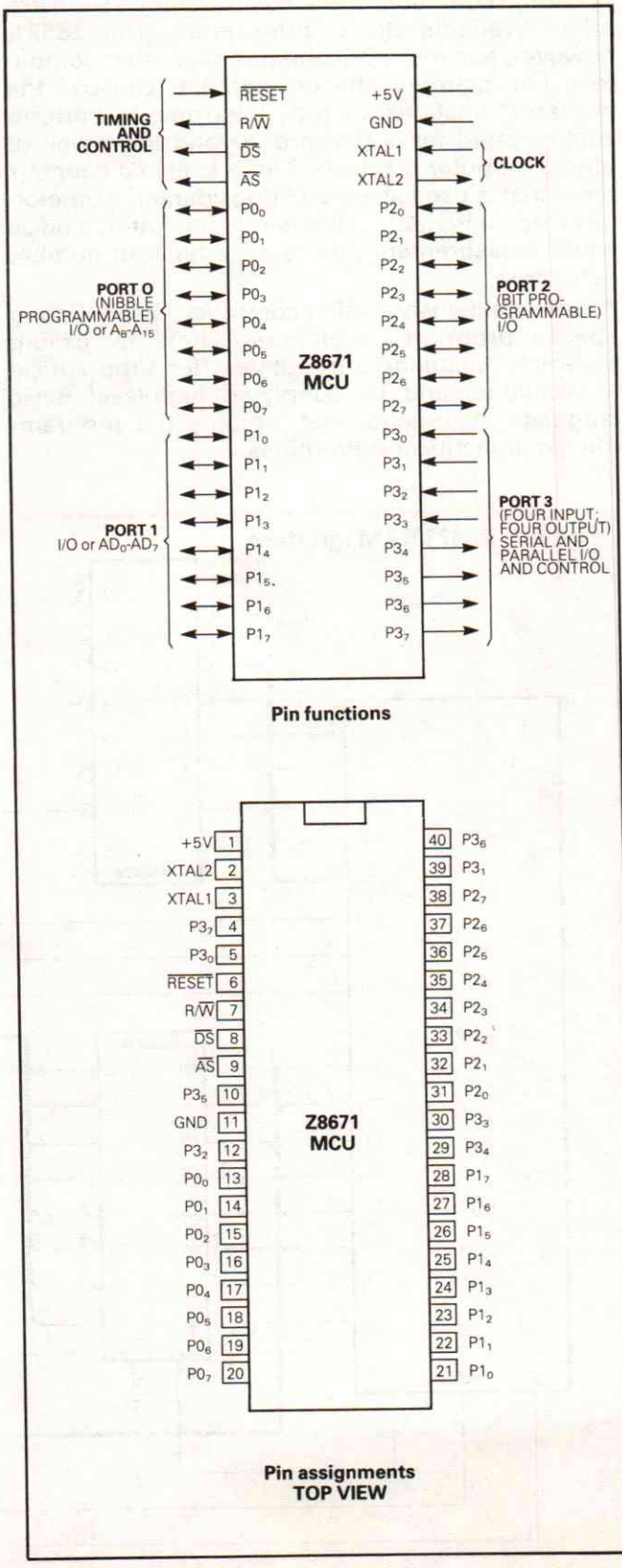
Basic/Debug is a subset of Dartmouth Basic, which interprets Basic statements and executes assembly language programs located in memory. Basic/Debug can implement all the Dartmouth Basic commands directly or indirectly.

One advantage to programming in Basic/Debug is the interactive programming approach realized because Basic/Debug is interpreted, not assembled or compiled. Modules are tested and debugged using the interactive monitor provided with Basic/Debug. Using Basic/Debug saves program development time by providing higher-level language statements that simplify program development. Using the INPUT and PRINT statements simplify debugging.

The Z8671 microcomputer

Basic/Debug controls the memory interface, serial port, and other housekeeping functions performed by the assembly language programmer.

The Z8671 uses ports 0 and 1 for communicating with external memory. Port 1 provides the multiplexed address/data lines (AD_0-AD_7); port 0 supplies the upper address bits (A_8-A_{15}). The Z8671



also uses the serial communications port for communicating with a terminal. Serial communication takes two pins from port 3, leaving six I/O pins from port 3 available to the user. The serial communication interface uses one of the two counter/timers on the Z8671 chip.

All other functions and features on the Z8601 are available with the Z8671. The user may reconfigure the Z8671 in software as a Z8601 if desired.

Applying the Z8671

Applications of the Z8671 range from a low-complexity home microcomputer that is memory intensive to an inexpensive, I/O-oriented microcontroller.

For home computer users, Basic/Debug is used like other available Basic interpreters. The Z8671, however, has many advantages over other computers. For example, the programmer can use the available functions such as interrupts to perform sophisticated tasks that are beyond the scope of other computer products. There is also a counter/timer that is used as a watchdog counter, a time-of-day clock, a variable pulse width generator, a pulse width measurement device, or a random number generator.

As an inexpensive microcontroller, Basic/Debug speeds program development time by calling assembly language subroutines (for time critical applications) and by supplying high-level Basic language statements that simplify the programming of noncritical subroutines.

Architecture

Two major design goals were set for this Z8671 Basic board. First, the board was to be simple. Second, the board needed to allow the user to write Basic programs and to utilize the features of the Z8601.

Overview

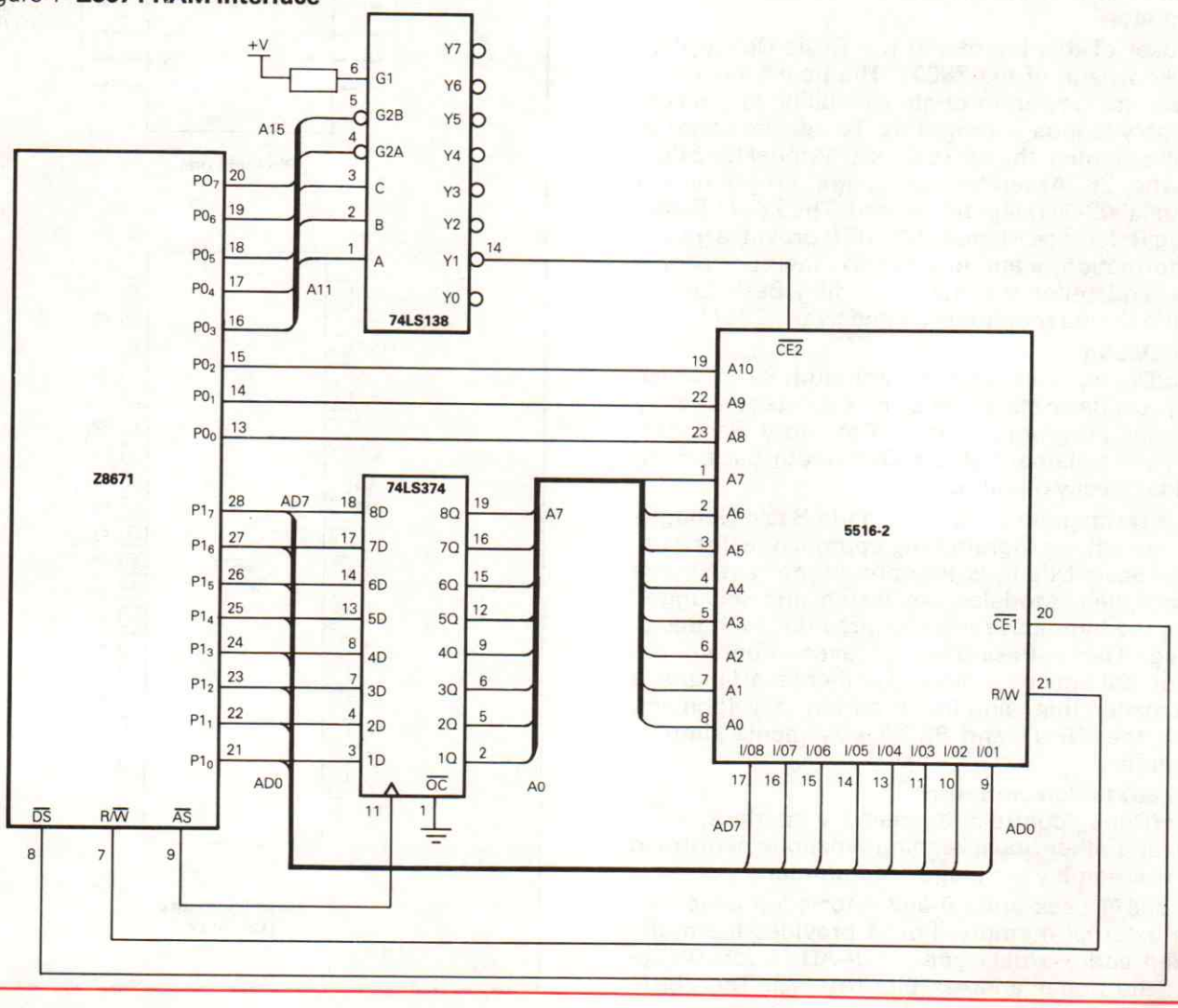
The board has eight IC packages:

* Z8671 (Z8601 preprogrammed with Basic/Debug)
* 5516 (2K bytes of static RAM) or 6264 (8K bytes of static RAM)
* 2732 (4K bytes of EPROM)
* 1488 (RS232 line driver)
* 1489 (RS232 line receiver)
* 74LS00 or 74LS10 (NAND gate package)
* 74LS138 (3 to 8 line decoder)
* 74LS374 (Octal Latch).

With these chips, a complete microcomputer system can be built with the following features:

* 2K byte Basic/Debug interpreter in the internal ROM.
* 2K or 8K bytes of user RAM
* 4K bytes of user-programmable EPROM
* Full-duplex serial operation with programmable baud rates
* RS232 interface
* 8-bit counter/timer with associated 6-bit prescalers
* 124 general-purpose registers internal to the Z8671

Figure 1 **Z8671 RAM interface**



- ★ 14 I/O lines available to the user
- ★ 3 lines for external interrupts
- ★ 3 sources of internal interrupts
- ★ Sophisticated, vectored interrupt structure with programmable priority levels. Each can be individually enabled or disabled, and all interrupts can be globally enabled or disabled
- ★ External memory expansion up to 124K bytes
- ★ Memory-mapped I/O capabilities.

This microcomputer can be used as a microcontroller, in which case a terminal is attached, via the RS232 interface, and Basic/Debug is used to create, test, and debug the system. When the system is debugged, the program is put into the EPROM, the terminal disconnected, and the board run standing alone. The terminal can be reattached at any time to monitor the subroutines running on the board.

The proposed boards meet the design requirements of simplicity and of allowing the user to write and debug programs in Basic while maintaining access to the Z8671 on-chip features.

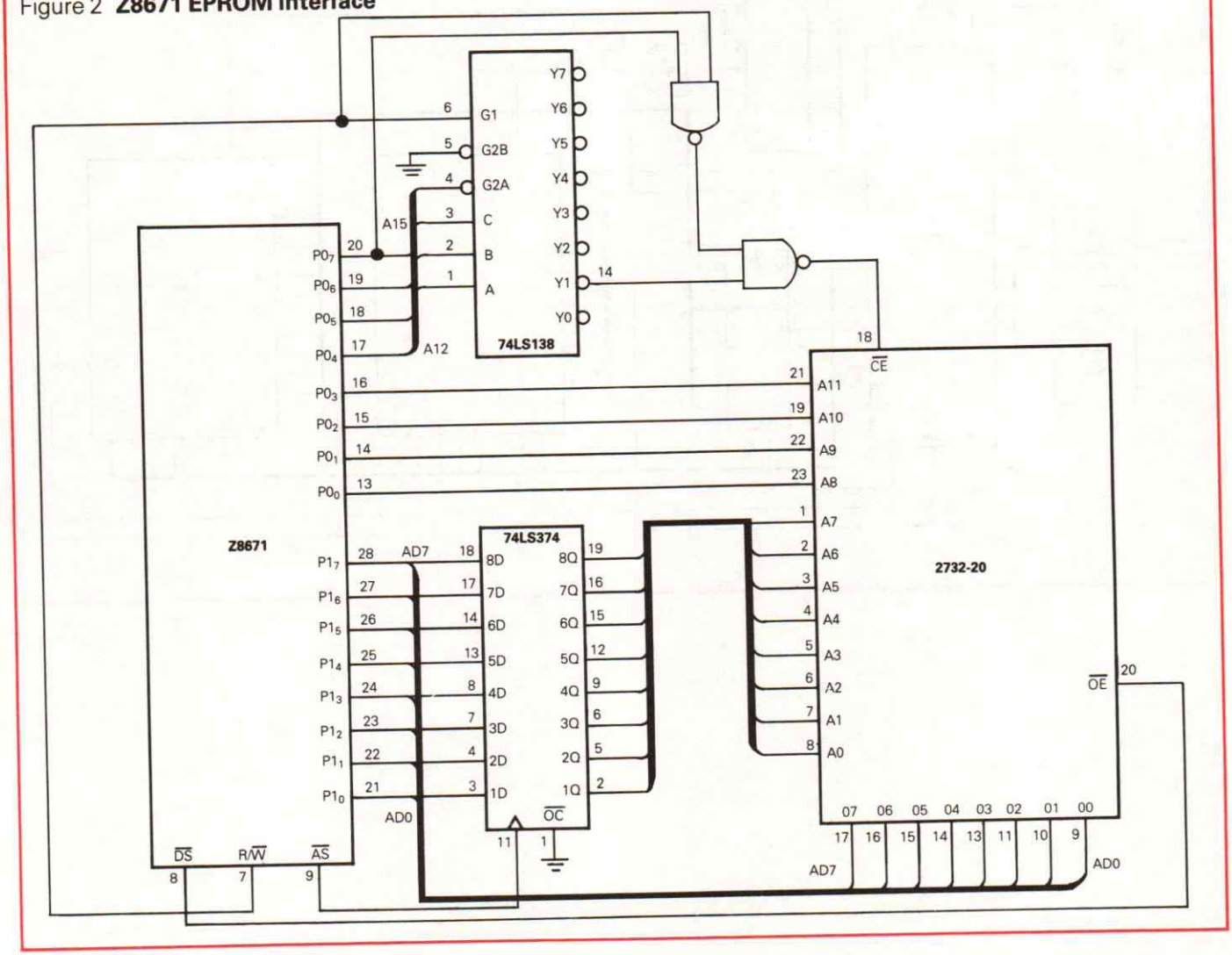
Interfacing the Z8671 with external memory

A simple RAM interface using the 5516 (2K byte) static Ram is shown in Figure 1, and a ROM interface using the 2732 (4K byte) EPROM in Figure 2. It is a fairly standard circuit with the 74LS374 being used to latch the low byte of the address and a 74LS138 providing address decoding. The major difference between the two is that the ROM circuit prevents memory accesses during a write cycle thus avoiding bus contention. This is carried out using the R/$\overline{W}$ signal as an enable on the address decoder. For a circuit using ROM and RAM this method cannot be used because the address decoder is used for both ROM and RAM. Examples of a mixed circuit can be seen in the full board designs (Figures 4 and 5).

Mapping the ROM at address 1000H to 1FFFH maintains compatability with Basic/Debug's Auto Start-up procedure. If this feature is not required then the address decoder can be omitted and $\overline{CE}$ can be obtained from the high order address bits.

Unique address space decoding can also be omitted for the RAM circuit but the Z8671, which sizes the RAM on power-up, could think that it has more RAM than there really is. This could cause problems with the GOSUB stack corrupting the Basic program and so care must be taken if a large Basic program is being run from RAM.

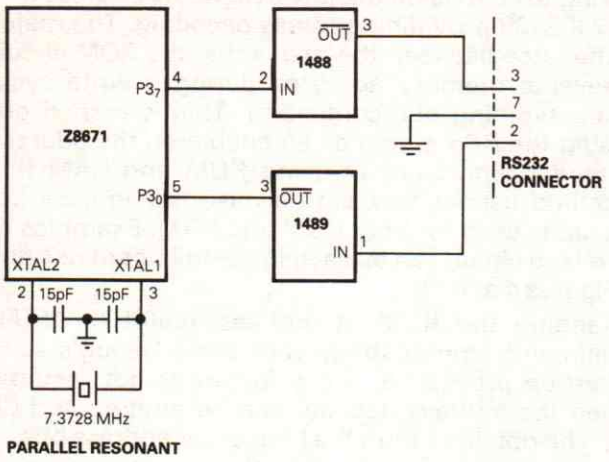
Figure 2 **Z8671 EPROM interface**



Interfacing the Z8671 with RS232 port

The Z8671 uses its serial communication port to communicate with the RS232 port. Driver and receiver circuits are required to supply the proper signals to the RS232 interface. The circuit of Figure 3 shows the interface between the Z8671 and the 1488 and 1489 for serial communication via the RS232 interface.

Figure 3 **Z8671 Interface for serial communications**



The serial interface is a simple type using only SI, SO and GND. The control signals CTS, DSR etc are therefore not required.

The Z8671 uses one timer and its associated prescaler for baud rate control. When the Z8671 is reset, it reads location FFFD and uses the byte stored there to select the baud rate. The boards described in this application note use EPROM to select the baud rate. On reset, the Z8671 reads FFFD, which is in the EPROM, and decodes the baud rate from the contents of that location. The baud rate can be changed in software.

Figures 4 and 5 show the full board designs implemented for this application note.

Figure 4 **Z8 Basic/Debug system with 4K EPROM and 2K RAM**

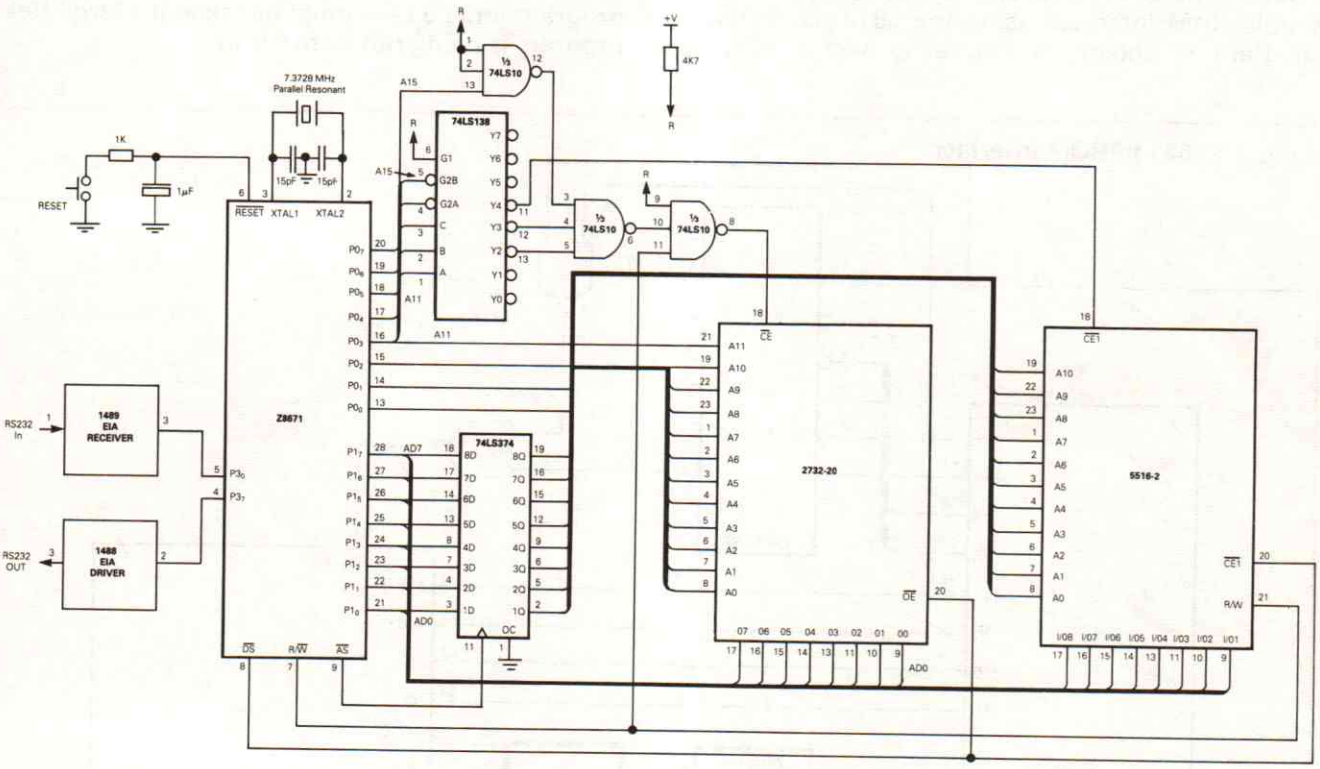
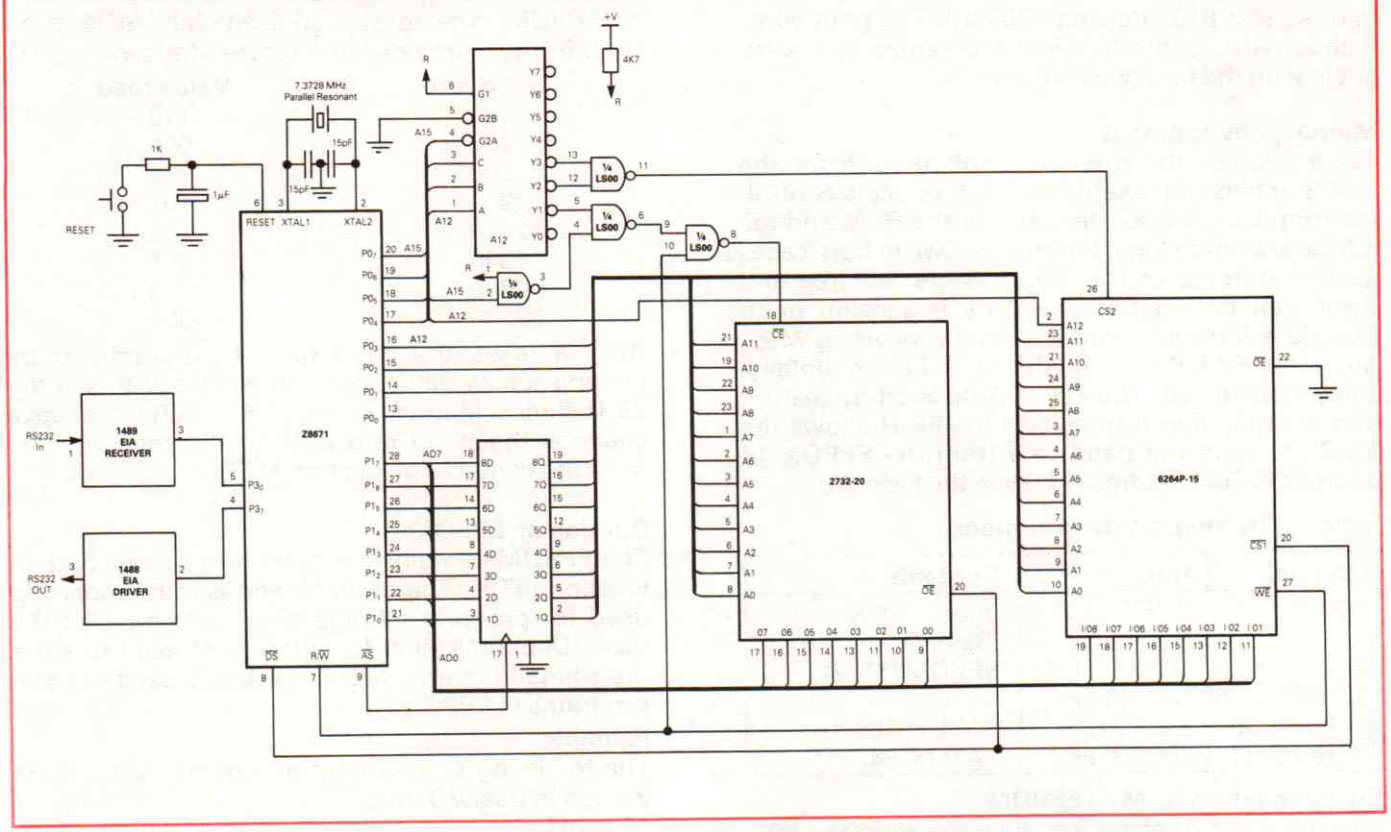


Figure 5 **Z8 Basic/debug system with 4K EPROM and 8K RAM**



Uncommitted I/O pins and other pins

Using the above design, port 2 is available for user applications. Any of the port 2 pins can be individually configured for input or output. There are also six pins in port 3 available to the user. The port 3 input pins can be used for interrupts.

Software

Getting started

The Z8671 board needs +5V and ground to run all components on the board except the 1488 EIA line driver. The 1488 needs +12V and −12V in addition to ground. (If not using terminal, the EIA driver/receiver circuit is disconnected. Consequently, the +12V and −12V lines are not required.)

The RS232 port can interface to any ASCII Terminal if the baud rate setting is matched to the value programmed into the EPROM. With power supplied to the board and the terminal connected to it, the reset button resets the Z8671 and the prompt character appears (":").

The board is ready for a Basic command when the ":" appears. The following sequence is a simple I/O example:

```
:10 input a
:20 "a=";a
:run
?5
a=5
:list
10 input a
20 "a=";a
:
```

When a number is entered as the first character of a line, the Basic monitor stores the line as part of a program. In this example, "10 input a" is entered. Basic stores this instruction in memory and prints another ":" prompt. The Run command causes execution of the stored program. In this example, Basic asked for input by printing "?". A number (5) is typed at the terminal. Basic accepts the number, stores it in the variable "a", and executes the next instruction. The next instruction (20 "a=";a) is an implied print statement; writing an actual "print" command is not necessary here. This line of code produced the output "a=5". The command "list" caused Basic to display the program stored in memory on the terminal.

Reading directly from memory

Basic lets the user directly read any byte or word in memory using the Print command and "@" for byte references or " ↑ " for word references:

```
:print @8
32
:printhex (@8)
20
:printhex ( ↑ 8)
2000
:
```

The first statement prints the decimal value of Register 8. The next statement prints the hexadecimal value of Register 8 and the last statement prints the hexadecimal value of Register 8 (20H) and Register 9 (00H).

Writing directly to memory

Basic lets the user write directly to any register or RAM location in memory using the Let command and either "@" or " ↑ ".

```
:@% a=%ff
: ↑ 8192=255
:print @10
255
:printhex ( ↑ %2000)
FF
:
```

The Let command is implied to save memory space but can be included. The first statement loads the

hexadecimal value FF into register 10 decimal (AH). The next instruction loads the decimal value 255 into register 8192 decimal (2000H). The print commands write to the terminal the values that were put in with the first two instructions.

Memory environment

Table 1 gives the memory configuration for the Z8671 application examples. Chip select is controlled from the address decoder for the RAM and the address/decode logic for the EPROM. In both cases several outputs of the 74LS138 are left free and these can be used for memory expansion or to decode addresses from peripheral devices. Mapping the EPROM at 1000H to 1FFFH maintains compatability with the Z8671 Auto Start-up feature and the mapping from 8000H to FFFFH allows the Z8671 to read the baud rate from the EPROM at address FFFDH (address FFDH in the EPROM).

Table 1 **The memory environment**

Decimal	Hex	Contents
0-2047	0-7FF	Internal ROM (Basic/Debug)
4098-8191 & 32768-65535	1000-1FFF & 8000-FFFF	EPROM (2732)
8192-10239	2000-27FF	RAM (5516)
8192-16383	2000-3FFF	RAM (6264)

Switching from RAM to EPROM

Register 8 and Register 9 contain the address of the first byte of a user program or, if there is no program, the address where the Z8671 will put the first byte of a user program. In this application example, when the Z8671 is reset and the EPROM does not contain the address, Register 8 and Register 9 contains a Basic program which points into RAM. EPROM is selected by changing the contents of Register 8 from 20H to 10H (see Table 2).

Table 2 **The registers**

Decimal	Hex	Contents
22-23	16-17	Current Line Number
8-9	8-9	Address of the First Byte of User Program

For more details on the register assignments, refer to the Pointer Registers-RAM System section of the Z8 Basic/Debug Software Reference Manual.

After the instruction " ↑8=%1000" is executed, the Z8671 accesses the EPROM on the Basic/Debug Board.

The example below shows how to switch from RAM to EPROM. The example uses two separate programs, one in RAM and one in EPROM. The RAM program is listed first, then the EPROM.

```
:printhex ( ↑8)
2000
:list
10 "executing out of RAM"
: ↑8=%1000
:printhex ( ↑8)
1000
:list
10 "executing out of EPROM"
:
```

Baud control

The baud rate is selected automatically by reading location FFFDH and decoding the contents of that location when the Z8671 is reset (the Z8 Basic/Debug Software Reference Manual contains the baud rate switch settings in Appendix B). This application example holds the baud rate settings in its EPROM. The least significant bits of location FFFD hex will provide baud rates as follows:

Baud rate	Value read
110	110
150	000
300	111
1200	101
2400	100
4800	011
9600	010
19200	001

After a reset, the baud rate is programmed by loading a new value into counter/timer 0 (see the Z8 Technical Manual, section 1.5.7). A Reset always changes the baud rate back to the rate selected from the contents of location FFFD.

Burning an EPROM

The EPROM contains the baud rate section byte in location FFDH. The other locations in memory are used for program storage. See section 6.3 of the Basic/Debug Manual for the format used to store programs in memory. This format is used to store programs in EPROM.

Example

The following is a printout of a game Super Brain written in Basic/Debug.

```
10 @243=7
20 @242=10
30 @241=14
40 x=usr(84): a=@242-1: x=usr(84): b=@242-1
50 x=usr(84): c=@242-1: x=usr(84): d=@242-1
55 "":i=0
100 "guess ",: in e,f,g,h
110 i=i+1
300 j=%7f22: k=%7f2a
301 t=0
302 r=0: p=0
310 if ↑j= ↑kp=p+1
320 j=j+2: k=k+2: t=t+1: if4>t310
330 j=%7f22: k=%7f2a
331 t=0
340 if ↑j= ↑kr=r+10: ↑j= ↑j+10: l=3
341 j=j+2
350 t=t+1: if 4>t310
351 j=%7f22
352 t=0
360 k=k+2: if%7f31>k340
363 j=%7f22: k=%7f2a
366 if ↑j>9 ↑j= ↑j-10
367 j=j+2
368 if%7f29>j366
370 "right";r; "place";p
380 if4>p100
390 y=999
400 "right in";i;"guesses; play another y/n":
    input x
410 if x=y10
```

Lines 10 through 50 comprise the random number generator for the program. The three lines:

```
10 @243=7
20 @242=10
30 @241=14
```

initialize counter/timer 1 to operate in modulo-10 count. Refer to the Z8 Technical Manual for complete information on initializing timers.

The "user (84)" function waits for keyboard input, the ASCII value of the key is returned in a variable

with the following command:

```
:10 x=usr(84): ""
:15 printhex(x)
:run
5
35
:
```

In the above example, the program waits at line 10 until keyboard input, in this case the number 5. The input value is stored in ASCII format in the variable "x". The line:

40 x=usr(84): a=@242−1: x=usr(84): b=@242−1

waits for input, reads the current value of timer 1, subtracts 1 (to get a number between 0 and 9), and stores the number in variable a. Then it waits for keyboard input at the second user function call, reads the current value of timer 1, subtracts 1, and stores the number in variable b. Line 50 of the example program gets two more random numbers and stores them in variables c and d. The four-digit random number is located in variables a,b,c, and d.

Line 300 assigns the location of variable a to variable j and the location of variable e (the first variable in the guess string) to the variable k. The strategy is to access these variables indirectly and to increment pointers j and k to access the variables.

A colon is used to separate commands on the same line. This is useful in packing the program into a small amount of memory space. The code, however, is harder to read. See section 5 of the Basic/Debug manual for more information on memory packing techniques.

Below is a sample run of the Mastermind Program:

```
:run
(<RETURN> on the keyboard is entered four times
  here)
guess ?0,1,2,3
right 2 place 0
guess ?4,5,6,7
right 2 place 1
guess ?0,2,4,6
right 3 place 2
guess ?4,2,1,6
right 4 place 4
right in 4 guesses
play another? y/n
?n
:
```

Conclusion

The design of these application examples met the major design goals of simplicity and functionality. The first goal is accomplished by prudent selection of support components, excluding any unnecessary chips. The board allows the user to exercise the full power and flexibility of the Z8601 not used by Basic/Debug. The user can write and debug Basic programs without detailed knowledge of the Z8601.

The Basic application example demonstrates a memory interface that is applicable for all Z8 Family members.

The software section explains the memory environment and gives several examples of Basic/Debug. These examples are a good introduction to the board and to Basic/Debug.

The Z8671 is a customized extension of the Z8601 single-chip microcomputer. The simplicity of the Basic application example demonstrates the flexibility of the Z8601 microcomputer in an expanded memory environment.

RS data

R.S. Components Ltd. PO Box 99, Corby, Northants, NN17 9RS
An Electrocomponents Group Company
Telephone: 0536 201234
©RS Components Ltd. 1984